小马哥的 Java 项目实战营

Java EE 项目 - 第八节 持续集成和持续交付

小马哥（mercyblitz）

我是谁?

小马哥（mercyblitz）

- 父亲
- Java 劝退师
- Apache Dubbo PMC
- Spring Cloud Alibaba 架构师
- 《Spring Boot 编程思想》作者

议题

- 持续集成
- 持续交付
- CI/CD 工具
- Jenkins 运用和管理
- 问答互动

持续集成

- 基本概念

一种软体工程流程，是将所有软件工程师对于软体的工作副本持续整合到共用主线（mainline）的一种举措。该名称最早由葛来迪·布区（Grady Booch）在他的布区方法中提出，在测试驱动开发（TDD）的作法中，通常还会搭配自动单元测试。持续整合的提出主要是为解决软体进行系统整合时面临的各项问题，极限编程称这些问题为集成地狱（integration hell）。

持续集成

- 理论基础

持续整合的宗旨是避免整合问题，如同在极限编程(XP)方法学中描述的整合地狱。持续整合并非普遍接受是用来改善整合频率的方法，因此重要的是区分两者所带来的效益。

在极限编程方法学，持续整合需要达到最佳成果，必须依靠著自动化整合单元测试并通过测试驱动开发。首先必须设想在上线运作之前，已在开发环境完成并通过所有的单元测试。这将帮助避免一个开发者的作业流程，导致其他开发者作业的中断。如果有需要，可以在完整上线运作之前进用部分已完成的功能，例如使用功能切换。

持续集成

- 理论基础

接著进行CI伺服器建置概念的阐述、自动化执行单元测试的周期与每次测试需要提交给开发者的报告。建置CI伺服器的用途(不一定要执行单元测试) 已经开始在极限编程(XP)社群之外的团队练习。如今，许多企业组织已经开始采用持续性整合，而非采用完整的极限编程(XP)。

除了自动化单元测试，组织在运用持续性整合(CI)一般会建置CI伺服器来维护持续性套用品质控制的程序-小部分的影响，并且经常性使用。除了执行单元与整合测试之外，还有额外的静态与动态测试，量测与描述效能，从程式来源码摘录与文件格式与促成手动品质保证(QA)程序。

持续集成

- 理论基础

持续性品质控制应用程式用意在提升软体品质以及减少交付的时间，在完成所有开发后，取代传统软体上线品质控制机制。此非常相似进行频繁整合的最初概念让整合得以在QA程序上更容易地达成。

同样的道理，持续性交付的最佳实践进一步扩展了持续性整合(CI)，以确保软体检核在主要程序上并且能够布署到使用者以确保实际的布署流程可以非常快速。

持续集成

- 工作流程

当从事变更时，开发者会从目前基础程式码库复制以进行作业，其他开发者提交程式码的变更至来源程式码库，并透过副本的方式取代来源程式码库的程式码。不只变更目前的程式码库，新的程式码也可以新增成为程式库、其它共用资源与潜在冲突。

当分支程式码保持在取出状态时间越长，当分支程式码开发者进行主线重新整合时，就愈容易遭遇整合多重冲突的风险以及失败。当开发者将程式码提交到程式码库时，首先必须更新程式码以反映他们在程式码库中的更改，因为他们拿到了副本。程式码库包含的更改越多，开发人员在提交自己的更改前必须执行的工作越多。

持续集成

- 工作流程

终于，该程式库也许变成非常不同于开发者的目标程式码，他们进入有时候被称为合并地狱或整合地狱的阶段，这时候开发者所花费的整合时间，将超过最初程式码开发的时间。

持续性整合涉及预先整合与预先与经常性的整合，借此来避免踩到整合地狱的陷阱，实践的目标是减少重工、减少成本与时间。

持续性整合补充的实践是在提交成果之前，每个开发人员必须执行一个完整的构建与执行及通过所有的单元测试、整合测试，这些都是当持续性整合伺服器侦测到程式码有新的提交时，必须经常性与自动化的进行。

持续集成

- 历史

葛来迪·布区于1994年出版的《Object-Oriented Analysis and Design with Applications》第二版中，首次提出持续整合这个名词。

1997年，肯特·贝克与Ron Jeffries建立了极限编程方法，将持续整合作为极限编程的一部份。

持续集成

- 最佳实践

持续性整合－经常将新的或改变的程式码与现有的程式码库进行汇集，这作业应该频繁地发生，在提交和构建之间不存在中间窗口，并且没有错误可以在没有开发人员注意到并立即纠正的情况下产生。最佳的做法是透过每次提交一个程式库来触发建构，而不是定期预定的版本才进行建构。在快速提交的多开发者环境实践是这样的：在每次提交之后的短时间内触发，然后在时间到期后开始建构，或者在上次建构间隔一段时间之后。许多自动化工具都有提供相关的自动化排程。另一个要因是建构一个支援原子提交的版本控制系统，此系统可以让开发人员的每次变更都可成为单一提交操作，但如试图从只有改变一半的档案进行构建没有意义。

持续集成

- 最佳实践
  - 维护一个代码库

这种做法意味著使用专案来源程式码版本控制系统。所有专案相关的程式码都需要储存在该程式码库中，在这种做法的控制界中，依惯例该系统应该可以从新取出的进行构建并且不需要额外的作业。提倡极限编程法的马丁・福勒也主张，必须有简单的工具来支援程式开发的分支作业。相反的，最好将所有变更整合起来而不是同时维护多个版本的软体。简单的说，这是将软体开发工作版本化的地方。

持续集成

- 最佳实践
  - 自动构建

透过一个单一指令来达成系统建构。许多的建构工具软体如MAKE已存在许多年，其他较新的工具程式都频繁的使用在持续性整合环境。建构的自动化应该包含自动化作业与整合作业，并且通常包含正式环境的布署。在许多案例中，程式码的建构不仅仅只是编译二进位元，通常总是伴随的文件的产生、网站的建构、状态数据与布署的封装媒体。(如Debian的DEB、Red Hat的RPM与微软的MSI档案)。

持续集成

- 最佳实践
  - 让构建时会自我测试

一旦代码编辑好，下一个阶段应该要进行所有的测试，以确保软体开发的成果符合预期。

  - 每人每天都应提交一次

透过定期的承诺，每个提交者都能够减少变更冲突的数量。一次检查一个星期的工作成果时，遭遇到整合冲突的风险相对交高，这时排解的困难性也相对升高。早期性系统的局部冲突，将可以让系统团队及时因应与进行调整建构的方向。

一天至少提交一次(每个功能构建一次)通常被定义为持续整合的一部分。此外，建议每晚在提交之后立即进行建构，以上都是下限值，实际上的频率往往要高出许多。

持续集成

- 最佳实践
  - 每份提交都应进行建置

系统应该要在每次提交之后，针对当下的版本进行建构以确认程式可正确的整合。通常实务上会使用自动化进行持续整合，也许这个也可以手动进行。在许多时候，持续整合是使用自动化持续整合的代名词，透过持续整合伺服器或应用程序监视版本控制系统的变化，然后自动进行建构的过程。

  - 维持快速建置

每个建置必须要维持快速完成，如此一来便可以避免整合问题。

持续集成

- 最佳实践
  - 用线上环境的复本测试

拥有一个测试环境并不能保证一切顺利，也会在布署上线时产生错误，因为测试环境和正式环境或许存在很大的差距。然而，如果要建置一份与线上环境一模一样的测试环境，还需要成本考量。相反的，测试环境或是独立的预备环境应该要建置实际正式环境的扩展版本，以在可容许的成本内同时达到维护堆叠结构技术与细微化。在那些测试环境中，服务虚拟化是通常运来获得，随需求存取超出团队控制的依赖关系(如API、第三方应用程式、第三方服务与大型主机系统)、持续演变或者因太复杂无法在虚拟实验室中还原的情境。

持续集成

- 最佳实践
  - 让取得最新发布版本更容易

使建置容易让利益关系人与测试者使用，能够减少许多因为建置的成果不合乎需求的重工状况。此外，提早测是能够在提早程式布署前知道变更的缺陷。在某些情况下，也可以提前查出错误，从而减少解决问题所需的工作量。所有程式设计师都应该从储存库更新项目来开始新的一天。这样，他们将可保持该程式储存库的最新状态。

  - 任何人都可以检视最后建置的结果

系统应该要让任何人都可以容易寻找出建构是否中断，并且可以显示何人正在变更相关程式。

持续集成

- 最佳实践
  - 自动部署

大部分的持续整合系统允许在建置完成后自动执行程式码。因此能够写一段程式码来布署应用程式至任何人都可以观察的测试伺服器。在持续性整合未来的思考发展成像持续性布署迈进。持续性布署将要求直接将软体布署至测试环境中，这通常需要额外的自动化机制来防止程式缺陷。

持续集成

- 成本与效益

持续整合目的在产生以下效益：

- 及早发现整合错误，且由于修订的内容较小所以易于追踪，这可以节省专案的时间与成本。
- 避免发布日期的前一分钟发生混乱，当每个人都会尝试为他们所造成的那一点点不相容的版本做检查。
- 当单元测试失败或发生错误，若开发人员需要在不除错的情况下还原程式码库到一个没有问题的状态，只需要放弃一小部份的更改 (因为整合的次数频繁)。
- 让 "最新" 的程式可保持可用的状态供测试、展示或发布用。
- 频繁的提交程式码会促使开发人员建立模组化，低复杂性的程式码。

持续集成

- 成本与效益

关于持续性自动化测试的效益：

- 强制执行频繁的自动化测试纪律

- 当改变对全系统造成影响时立即反馈

- 自动化测试和持续性整合产生的软件度量(如程式码覆盖度量，程式码复杂度和功能完整性等)标准将开发人员集中在开发功能性，高品质的程式码上，并帮助开发团队发展。

持续集成

- 成本与效益

  持续整合的缺点包含：

  - 构建一个自动化测试套件需要大量的工作，包括不断努力以覆盖新功能，并依照特定情境进行程式码修改。
  - 建置系统需要一些工作，而且可能变得复杂，难以灵活修改。
  - 如果范围很小或包含无法测试的旧版程式码，持续性整合不一定有价值。
  - 增加的价值取决于测试的品质以及程式码的真实可测性。
  - 较大的团队意味著不断将程式码添加到整合队列中，因此追踪交付（同时保持品质）很困难，而排队可能会减慢所有人的进度。

持续集成

- 成本与效益

  持续整合的缺点包含：

  - 构建一个自动化测试套件需要大量的工作，包括不断努力以覆盖新功能，并依照特定情境进行程式码修改。
    - 无论是否采用持续性整合，测试被认为是软体开发的最佳实践，测试必须依循软体布署的最佳实务。自动化是诸如测试驱动开发之类项目方法的一个组成部分。
    - 持续性整合可以在不需要测试套件下执行，但是如果必须手动和经常地完成，生产产品的品质保证成本将会提高。
  - 建置系统需要一些工作，而且可能变得复杂，难以灵活修改。
  - 如果范围很小或包含无法测试的旧版程式码，持续性整合不一定有价值。

持续集成

- 成本与效益

  持续整合的缺点包含：

  - 增加的价值取决于测试的品质以及程式码的真实可测性。

  - 较大的团队意味著不断将程式码添加到整合队列中，因此追踪交付（同时保持品质）很困难，而排队可能会减慢所有人的进度。

  - 通过一天的多次提交和合并，功能的部分程式码可以轻松推送，如此一来整合测试将会失败直到整个功能开发完成。

持续交付

- 基本概念

一种软件工程手法，让软件产品的产出过程在一个短周期内完成，以保证软件可以稳定、持续的保持在随时可以释出的状况。它的目标在于让软件的建置、测试与释出变得更快以及更频繁。这种方式可以减少软件开发的成本与时间，减少风险。

持续交付

- 与DevOps的关系

持续交付与DevOps的含义很相似，所以经常被混淆。但是它们是不同的两个概念。DevOps的范围更广，它以文化变迁为中心，特别是软件交付过程所涉及的多个团队之间的合作（开发、运维、QA、管理部门等），并且将软件交付的过程自动化。另一方面，持续交付是一种自动化交付的手段，关注点在于将不同的过程集中起来，并且更快、更频繁地执行这些过程。因此，DevOps可以是持续交付的一个产物，持续交付直接汇入DevOps。

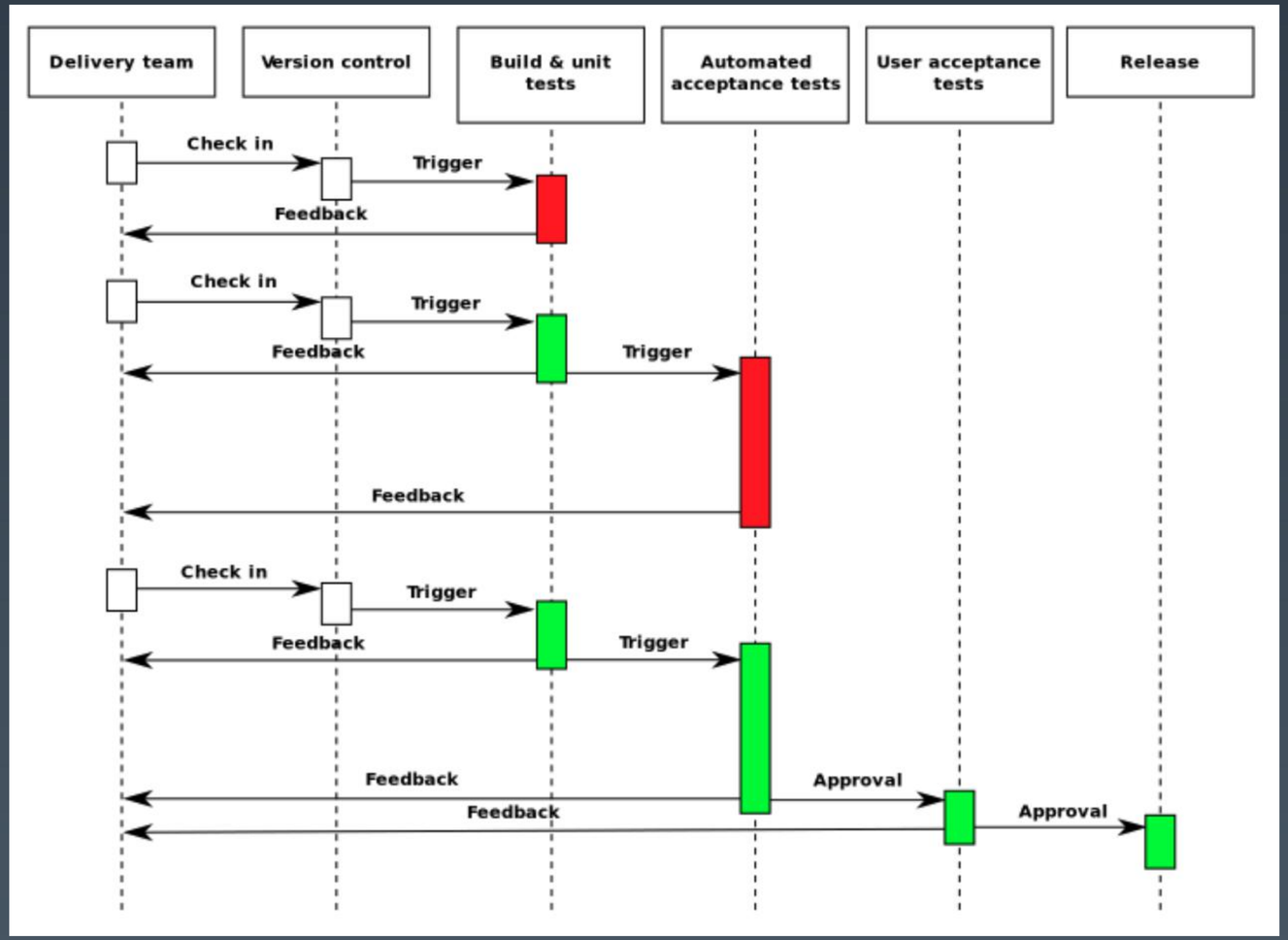
持续交付

- 与持续部署的关系

有时候，持续交付也与持续部署混淆。持续部署意味着所有的变更都会被自动部署到生产环境中。持续交付意味着所有的变更都可以被部署到生产环境中，但是出于业务考虑，可以选择不部署。如果要实施持续部署，必须先实施持续交付。

持续交付

- 原则


Delivery team
Version control
Build & unit tests
Automated acceptance tests
User acceptance tests
Release
Check in
Trigger
Feedback
Check in
Trigger
Feedback
Trigger
Feedback
Check in
Trigger
Feedback
Trigger
Feedback
Approval
Feedback
Approval

CI/CD 工具

- 流行工具
- Jenkins - https://www.jenkins.io/
- GitLab CI/CD - https://docs.gitlab.com/ee/ci/
- CircleCI - https://circleci.com/
- Github Build and Test

持续集成工具

- Jenkins（Hudson）

Jenkins是一款由Java编写的开源的持续集成工具。在与Oracle发生争执后，项目从Hudson项目复刻。

Jenkins提供了软件开发的持续集成服务。它运行在Servlet容器中（例如Apache Tomcat）。它支持软件配置管理（SCM）工具（包括AccuRev SCM、CVS、Subversion、Git、Perforce、Clearcase和RTC），可以执行基于Apache Ant和Apache Maven的项目，以及任意的Shell脚本和Windows批处理命令。Jenkins的主要开发者是川口耕介。Jenkins是在MIT许可证下发布的自由软件。

可以通过各种手段触发构建。例如提交给版本控制系统时被触发，也可以通过类似Cron的机制调度，也可以在其他的构建已经完成时，还可以通过一个特定的URL进行请求。

持续集成工具

- Jenkins 安装

Jenkins 提供多种安装方式，包括：

- Docker
- Kubernetes
- Linux
- Windows
- macOS
- WAR 文件

持续集成工具

- Jenkins 基础功能

Jenkins 支持不少的基础功能，包括：

- 安全认证
- 搜索
- 国际化
- 时区
- 远程控制 API
- Agent
- 插件整合

THANKS!